SPARCsystem 600MP VMEbus Implementation Guide

Sun Microsystems, Inc.
2550 Garcia Avenue
Mountain View, CA 94043
U.S.A.

Part No: 800-6738-11
Revision A, August 1992

Please
Recycle

Contents

Figures

Tables

Sun 600MP VMEbus Implementation Guide

Introduction

This *SPARCsystem 600MP VMEbus Implementation Guide*, P/N 800-6738-xx, provides VME integrators, OEMs, and Sun Sales personnel with technical details about the capabilities and the limitations of the SPARCsystem 600MP Series implementation. The following pages describe the VMEbus implementations for 630MP (5-slot), 670MP (12-slot), and 690MP (16-slot) systems.

This guide assumes a high level of technical familiarity with the subject. For a more general description of Sun VMEbus functionality and mechanics, refer to the *Sun-4/SPARC Hardware Configuration Guide*, FE295-0.

Reference Materials

The following reference materials may be helpful when using this manual:

- *The VMEbus Specification*, Rev C.1, Motorola, Inc.
- *Open Boot PROM 2.0 Toolkit Reference Guide*, P/N 800-6076-xx
- *Solaris 2.0 – Writing Device Drivers*, P/N 800-6502-xx
- *SBus Specification*, P/N 800-5922-xx

For a complete listing of the 600MP Series customer documentation, refer to the last section of this manual.

600MP Series VMEbus Features

The SPARCsystem 600MP system board VMEbus interface is designed to be as similar as possible to previous Sun VME interfaces.

The 600MP VMEbus has the following capabilities:

- uses VME C.1 specification base (same as Sun 4/300 family)

- any processor can access the VMEbus as a master and talk to any slave device
- the SPARCsystem 600MP acts as a slave device for VMEbus masters to perform DVMA accesses to system main memory
- any VMEbus master can access any SBus card through DVMA (however, an SBus DVMA master cannot access the VMEbus)
- the same, or improved, basic master/slave address and data capabilities as previous Sun VMEbus implementations
- I/O cache provides acceleration of sequential VME slave port activity similar to the Sun-4/400 and 3/400 Series products

600MP VME Subsystem Partition

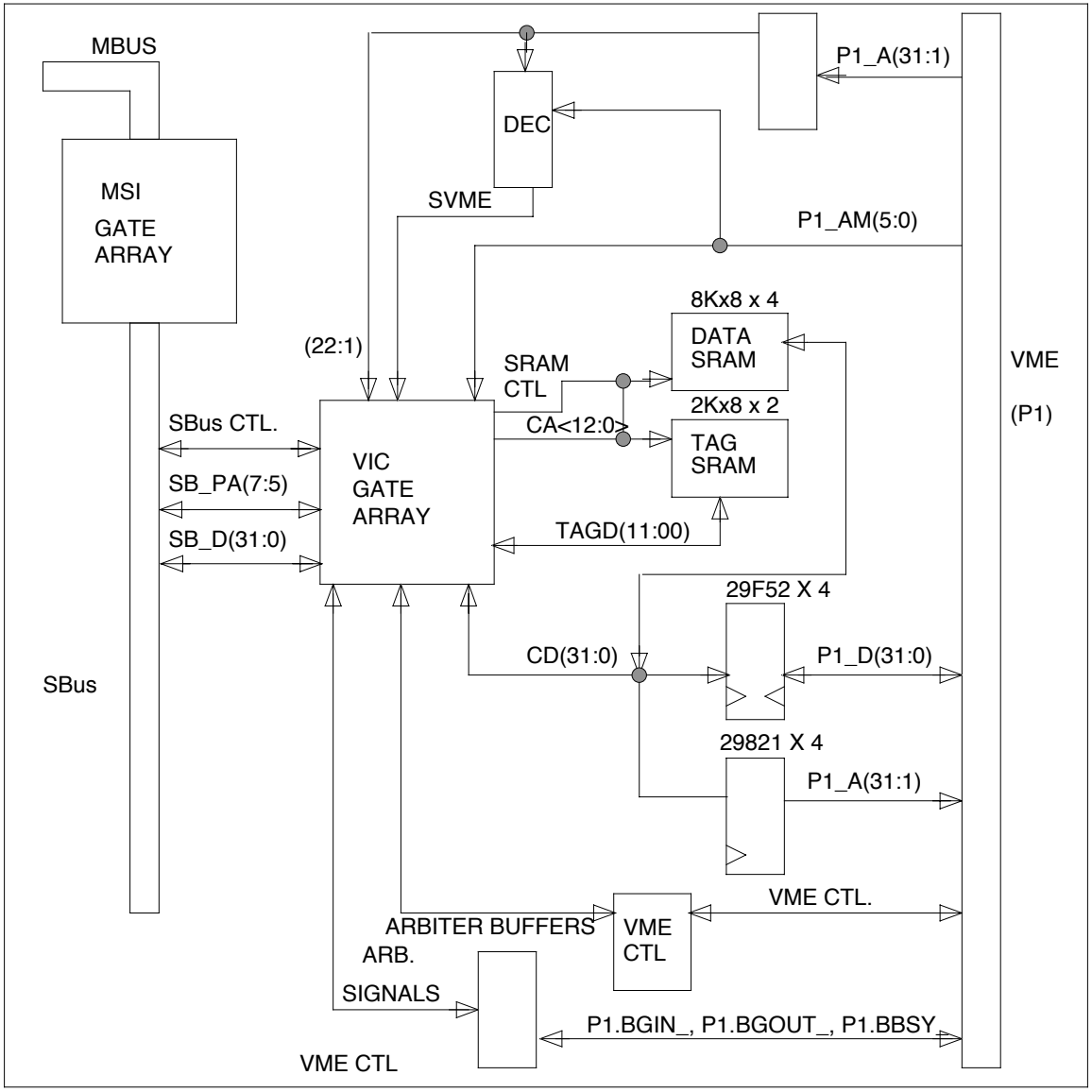


Figure 1 VME Subsystem Partition

Physical Address Space

Table 2 Physical Addresses (32-Bit Space)

PA(35:32)	32-Bit Space
0x0	Main Memory
0x1- 0x8	Reserved
0xA	VME Master Port, User, 16-bit Maximum Data
0xB	VME Master Port, User, 32-bit Maximum Data
0xC	VME Master Port, Supervisor, 16-bit Maximum Data
0xD	VME Master Port, Supervisor, 32-bit Maximum Data
0xE	SBus
0xF	Control Space

Table 3 Physical Addresses (Control Spaces)

PA(35:28)	Control Space
0xF0	Memory Control Space
0xF1- 0xFC	Reserved
0xFD	VME/IOC Control Space
0xFE	SBus/IOMMU Control Space
0xFF	System Space
PA(35:24)	
0xFF0	EPROM

Default Jumpers

Table 4 600MP Board Default Jumpers

Location	Default Installation	Label: Function
J1701	PIN (1&2) installed	VME ARB1: Enable Internal VME Arbiter; Factory set; do NOT change
J1702	PIN (1&2) installed	VME ARB2: Enable Internal VME Arbiter; Factory set; do NOT change
J1703	PIN (1&2) installed	VME ARB4: Enable Internal VME Arbiter; Factory set; do NOT change
J1704	PIN (1&2) installed	VME ARB3: Enable Internal VME Arbiter; Factory set; do NOT change
J1705	PIN (1&2) Not installed	ARBDIS: Disable Internal VME Arbiter; Factory set; do NOT change
J1706	PIN (1&2) Not installed	VMERST: VME Reset In
J1706	PIN (2&3) installed	VMERST: VME Reset Out
J1801	PIN (1&2) installed	VME17: Enable VME IRQ7 to System Board
J1801	PIN (3&4) installed	VME16: Enable VME IRQ6 to System Board
J1801	PIN (5&6) installed	VME15: Enable VME IRQ5 to System Board
J1801	PIN (7&8) installed	VME14: Enable VME IRQ4 to System Board
J1801	PIN (9&10) installed	VME13: Enable VME IRQ3 to System Board
J1801	PIN (11&12) installed	VME12: Enable VME IRQ2 to System Board
J1801	PIN (13&14) installed	VME11: Enable VME IRQ1 to System Board
J1801	PIN (15&16) Not installed	Not Used
J1802	PIN (1&2) installed	VMCK: Enable VME 16 MHz Clock (remove for In-Circuit Test)
J1803	PIN (1&2) installed	SLOT1: Enable VME slot-1 functionality

Maximum Performance

The performance goals for the Sun 600MP VME Master interface are:

- MVME Reads: 7.3 Mbytes/sec
- MVME Writes: 10.0 Mbytes/sec

The performance goals for the Sun 600MP VME Slave interface with the I/O Cache are:

- SVME Reads: 13.0 Mbytes/sec
- SVME Writes: 16.0 Mbytes/sec

The performance goals for the Sun 600MP VME Slave interface without the I/O Cache are:

- SVME Reads: 5.0 Mbytes/sec
- SVME Writes: 7.2 Mbytes/sec

Note – All calculations are for an *ideal* VME master or slave with no other bus traffic on the Sun 600MP system board. This is the theoretical upper limit. Whenever possible, take advantage of the I/O Cache. Without the I/O Cache, performance will be reduced.

Factors Affecting Latency

The following factors may affect latency adversely:

- number of processors in a system increases bus traffic
- number of VME boards being used
- memory refreshes will affect performance
- other boards plugged into the SBus will impact if doing DVMA
- amount of Ethernet traffic
- amount of SCSI disk activity

Slot Location

Note – In the following discussions, VME Slot 1 is assumed to contain the Sun 600MP System Board. In some configurations, this is actually Slot 4 of the backplane. Refer to section 1.11 of this manual for detailed descriptions.

For VME boards using the Sun 600MP system board Slave VME port, the closer the VME board is to slot 1 (CPU location), the higher the performance. The Single Level Arbiter naturally prioritizes which boards receive the best service.

The Sun 600MP Bus Grant line (P1_BG3) is a daisy chain. When the CPU grants the bus, the first board has the first opportunity to grab the bus; second board is next, and so on.

Position high-priority VME boards using DVMA as close to slot 1 as possible.

Timing

The Sun 600MP system board VMEbus is a fully asynchronous bus. All Sun 600MP VME internal sequencers, synchronizers, and synchronization operate at 20MHz. Asynchronous VME control signals are 2-clock synchronized with the 20MHz clock before being used by the internal state machines. The VME C.1 specification defines the timing parameters of setup and hold relationships for different signals, and the Sun 600MP System Board is designed to meet these timings.

Arbitration Details

The Sun VME interface contains an SGL (Single Level) arbiter. The arbiter function will only be enabled if the SLOT1_ pin is low. The SGL arbiter will only respond to requests on P1_BR3IN*. An external device on the CPU board will drive P1_BG(2:0)* high if the board is in slot 1.

Note – Using single level arbitration automatically causes geographical prioritization of VME boards. Boards which are closer to slot 1 receive better service than those further out. System integrators should bear this in mind and place critical devices as close to the CPU as possible. The SGL arbiter supports parallel arbitration to minimize the delay between bus masters.

The MVME interface will make use of the VMEbus requester. The bus requester will be ROR (release on request) and will always use level 3, which is highest priority. If SLOT1 is not true, the requester will need to get a bus grant from an arbiter which is in slot 1 of the VME bus.

Note – The Sun 600MP Master Port does not use address pipelining on the VMEbus.

If there is an error on a Master Read Cycle, it is reported synchronously to the processor. If there is an error on a Master Write Cycle, the error is reported as an asynchronous fault causing a CPU trap.

The Master Port has a one deep write buffer allowing the Sun 600MP VME interface to acknowledge cycles from the processor as quickly as possible. The 600MP system board does not have to wait while the target VME Slave Board being accessed is accepting the cycle. If, however, this cycle fails, an asynchronous fault will occur, the CPU will be interrupted, and the status and address of the cycle will be captured in an internal system register. The trap handler reads this information and determines that the cycle has failed.

The Master Port times out a VME access after 204.8 uSec. The timer does not start until the Sun 600MP Master Port has obtained the bus and asserted P1_AS*. After 204.8 uSec, the bus cycle is aborted by driving P1_BERR*, the bus error signal.

Table 5 Definition of VMEbus Master Port

Master Port	Definition
Address Sizes	A32, A24, A16
PA(31:00) = 0xFFFFxxxx is A16 space	
PA(31:00) = 0xFFxxxxxx is A24 space except for A16 space	
Data Sizes	D32, D16, D8(E0)
Interrup Handlert	IH(7-1), D8(0), jumper disable per interrupt
Bus Time-out	>200 uSec from master assertion of AS*
Bus Requester	SGL, ROR
Bus Arbiter	SGL, jumper disable
Address Modifiers	
Generated	0x39, 0x3D, 0x09, 0x0D, 0x29, 0x2D

Slave Port

Table 6 Definition of VMEbus Slave Port

Slave Port	Definition
Address Sizes	A32, A24
A32 responds to the lowest 8 MB of VME address space	
A24 responds to the lowest 1 MB of VME address space	
PA(31:00) = 0xFFxxxxxx is A24 space except for A16 space	
Data Sizes	D32, D16, D8(E0);UAT and Bursts not supported
Address Modifiers	0x39, 0x3A, 0x3A, 0x3D, 0x3E, 0x09, 0x0A, 0x0D, 0x0E

The Slave Port supports address pipelining per the VME Specification. There is no support for block transfers.

The SVME port responds to 8 Mbytes of VME space in A32 mode and 1 Mbyte of VME space in A24 Mode. As in previous SPARC /Sun-4 implementations, the VME space is moved to the top of virtual memory. In this case, A(31:23) equals 0xFF going in to the IOMMU.

Note – Software: The 8 Mbytes of A32 space is moved to the top 8 Mbytes of DVMA space, starting at the location (4 Gbytes minus 8 Mbytes). The 1 Mbyte of A24 space needs to coincide with the lowest 1 MByte of A32 space, so it is moved to the same starting location, (4 Gbytes minus 8 Mbytes). This differs from prior Sun implementations where VME starts at 1 Mbyte from the top, at location (4 Gbytes minus 1 Mbyte). Refer to the *IOMMU and IO Cache* subsection for a complete memory map.

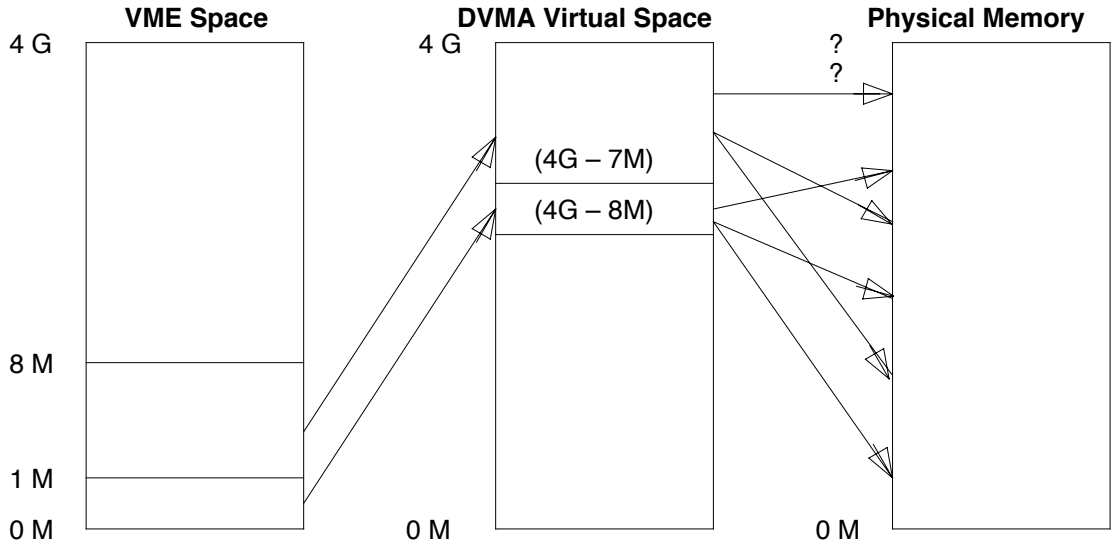


Figure 2 Overview of DVMA and VME Space Mapping

The Sun 600MP IOMMU is responsible for translating SBus DVMA addresses into a physical address in order to do the actual access. The CPU modules used in the Sun 600MP board use a 4K system page size. The IOC deals with 8K pages for the following reasons:

- NFS operations use 8K data buffers
- most disk transfers are multiples of 8K (56K or 64K)
- filesystem block size is 8K
- 8K page size matches the average I/O operation size

The IOC has only 1 32-byte block of SRAM for each 8K of VME space to allow multiple pages to use the cache at the same time. What this means is that the operating system must assign only EVEN numbers of 4K pages to VME devices that are cached. The cache is designed to give each I/O page its own cache line, and make line re-use very fast, since either line size matches the transfer size on the SBus. This provides efficient fill/flush properties and maximizes performance.

Space assigned should always begin and end on 8K boundaries. This should not be a problem, since existing Sun computers allocate VME space in multiples of 8K. The reason for this restriction is that an IOC block must not be shared by two different SVME masters at the same time. If, for instance, one master was reading while the other was writing, the IOC data could get corrupted. The IOMMU will still have one entry for each 4K page.

Note – Each pair of 4K physical pages that is logically associated with an 8K VME page must have the same state in the system cacheable bit, as well as the same write–access information.

I/O Cache

The Sun 600MP I/O Cache is provided to accelerate sequential VME slave port activity. The I/O Cache has no effect on Master VME activity. Unlike previous IOC designs at Sun, this IOC is not shared with SCSI or Ethernet traffic. IOC accesses to memory are always 32-byte bursts. On VME reads from main memory, leading fragments can be discarded. On VME writes to main memory that use the IOC, the writes always start and end on 32-byte boundaries independent of the addresses used.

Write-backs and flushes always trigger a 32-byte burst access. The IOC is not cycle-by-cycle coherent with main memory, but is coherent on a 32-byte burst basis. In SunOS this is known as the “Streaming I/O” model.

The IOC is a write-back cache with a no-write-allocate policy. Descriptors shared between the CPU and VME devices must be non-IOC cacheable. VME slave port accesses can use or bypass the IOC on an 8K page basis.

Each 32-byte line in the IOC will map to an 8KB section of VME address space; the mapping is a direct-map based upon VME A<22:13>. A total of 8MB of VME address space is allocated to the slave port, but only 1MB will respond in A24 space; the 1MB of A24 space overlays one of the 8MB of A32 space.

Due to the 8K mapping, IOMMU entries for VME must be made identically on a pair of sequential 4K pages; that is, the write-allowed, cacheable, and valid bits must be the same, although the physical pages do not have to be physically contiguous.

Table 7 I/O Cache Fields

Field	Description	Type	Bits
TAG	Identifies block within 8K page; compared to VME.A<12:05>	R	D (31:24)
V	Valid bit: Set when the tag contains valid information	W	D (23)
M	Modified: At least one byte of this cache line is dirty	R	D (22)
IC	IOC-cacheable: Set by the kernel during mapping. This is independent of system cacheability of the data.	W	D (21)
W	Write–allowed: Set by the kernel during mapping	W	D (20)
rsvd	reads as 0's, writing has no effect	R	D (19:0)

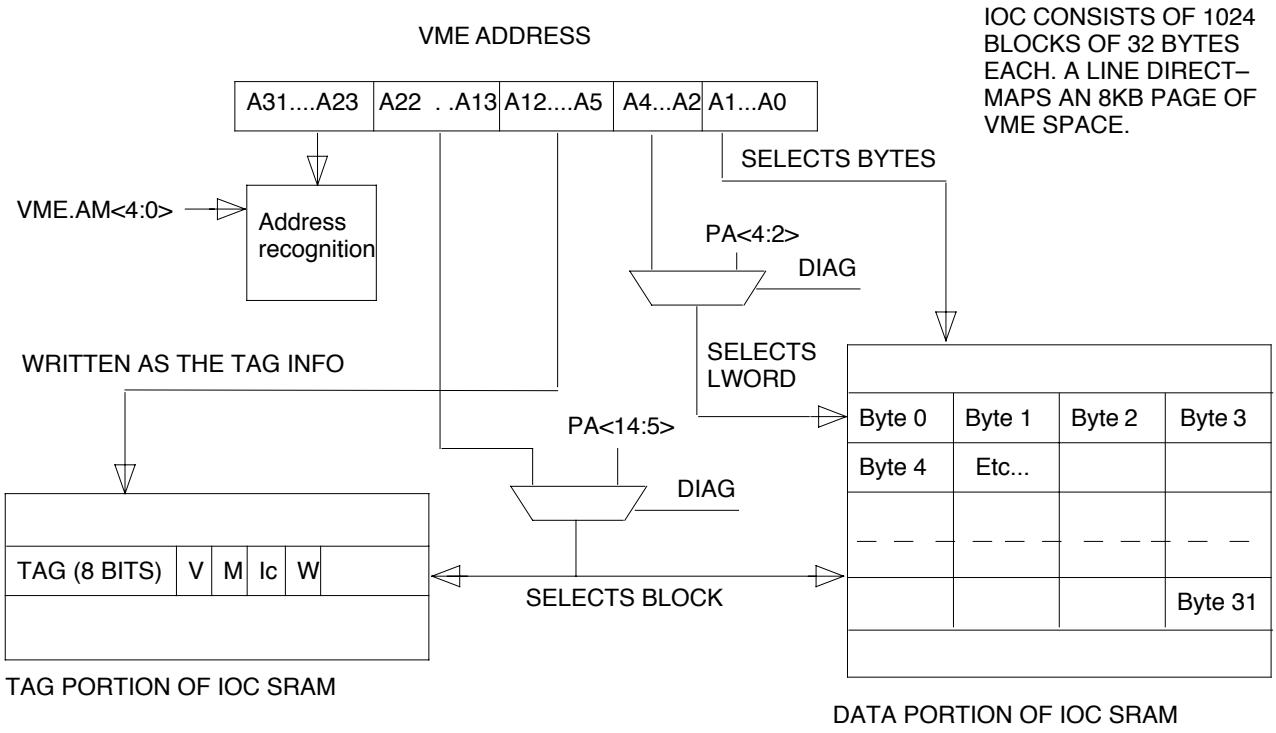


Figure 3 I/O Cache Data Block Diagram

Interrupts

The Sun 600MP Interrupt Handler is identical to previous Sun Interrupt Handlers, supporting 8 bit interrupt vectors. Interrupt levels are mapped to SPARC interrupt levels as shown in the following table.

Table 8 Sun 600MP Interrupt Handler

Level	Sources
1	SOFTINT.1
2	SOFTINT.2, VMEbus L1, SBus L1
3	SOFTINT.3, VMEbus L2, SBus L2
4	SOFTINT.4, on-board SCSI
5	SOFTINT.5, VMEbus L3, SBus L3
6	SOFTINT.6, on-board Ethernet
7	SOFTINT.7, VMEbus L4, SBus L4
8	SOFTINT.8, on-board Video
9	SOFTINT.9, VMEbus L5, SBus L5
10	SOFTINT.10, System Counter/Timer
11	SOFTINT.11, VMEbus L6, SBus L6, Floppy (PIO)
12	SOFTINT.12, Keyboard/Mouse, Serial Ports
13	SOFTINT.13, VMEbus L7, SBus L7, ISDN Audio (PIO)
14	SOFTINT.14, Per-processor Counter/Timer
15	SOFTINT.15, Asynchronous Errors (Broadcast)

The VME cycle is triggered by reading a pseudo register called the Vector Register.

The Sun 600MP board will not be a VME bus interrupter, so it will pass on any incoming IACKs. Modules can respond to VME interrupts by doing an odd–byte read to system control space. Refer to the *Address and Interrupt Vector Assignments* section of this manual for all internal VME control addresses.

The VIC chip decodes the control space access and issues the IACK_ cycle. The VIC chip considers the IACK signal to be just part of the VME addressing information and will treat the cycle like any VME bus read.

The VIC chip participates in the VMEbus IACK daisy chain by propagating P1_IACKIN_ when one of the VME data strobes is active.

Rev. C.1 Compliance

Variances from the VME Rev. C.1. Specification include the following:

- P1_ACFAIL* is asserted at the same time as P1_SYSRST*. It is not asserted at power out.
- Block Mode transfers are not supported by the SVME port. The VME AM code for BLT transfer is ignored, and the cycle will timeout. (Note: it is up to the other VME master to timeout these cycles.) Also, the MVME port will never generate BLT transfers.
- UAT (Unaligned transfers) are not supported by either MVME or SVME ports. The SVME port will generate P1_BERR* in response to a UAT cycle.
- All IACK cycles use 8–bit vectors. There is no support for 16–bit or 32–bit VME interrupt vectors.
- P1_BR3* is the only level request supported by the arbiter. The other levels are pulled high on the CPU board and will be ignored.
- The MVME port will not accept transfer sizes greater than 4 bytes from the SBus.
- If an MVME cycle times out on the VME bus, the VIC will drive the P1_BERR* low before withdrawing the cycle. (This is what the VME spec. requires, but previous Sun computers do not conform to this rule so it is worth mentioning.)
- The MVME port will only generate the following AM codes: 0D, 09, 2D, 29, 3D, 39. The SVME port will only recognize the following AM codes: 0D, 09, 0A, 0E, 3D, 39, 3A, 3E.

Connector Assignments

The following pages detail the backplane slot layouts and jumper locations for the x30 (5-slot), x70 (12-slot), and x90 (16-slot) systems.

SPARCsystem 630MP

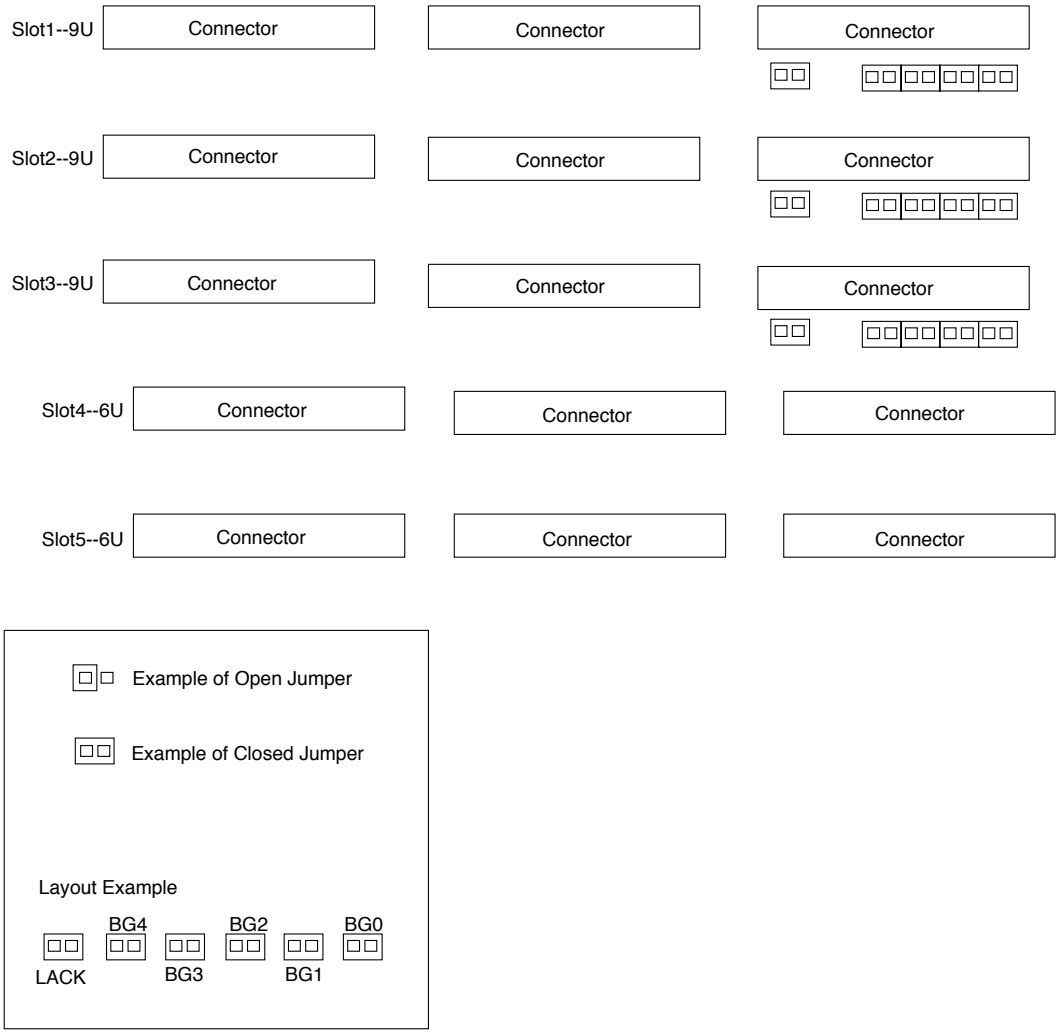


Figure 4 P Connectors for the SPARCsystem x30 Backplane

SPARCsystem 670MP

The first table below shows the x70 backplane jumper functions. Each jumper listed (left column) connects P1_BG0IN*-P1_BG3IN* to P1_BG0OUT*-P1_BG3OUT* and P1_IACKIN* to PA_IACKOUT* on the cardcage slot (right column). The second table shows the jumper locations on the backplane for each jumper function.

Bus termination is provided by the Clock jumpers which must be IN at locations P10, P11, P12, and P13. The +5v standby jumper must be in at P100.

Table 9 SPARCserver 670MP Backplane Jumper Functions

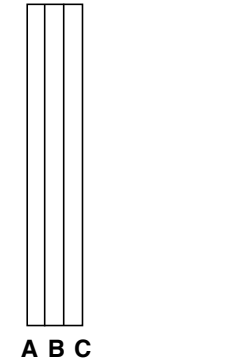
Jumper	Slot
P4XX	4
P5XX	5
P6XX	6
P7XX	7
P8XX	8
P9XX	9
P10XX	10
P11XX	11
P12XX	12

Table 10 SPARCserver 670MP Backplane Jumper Locations

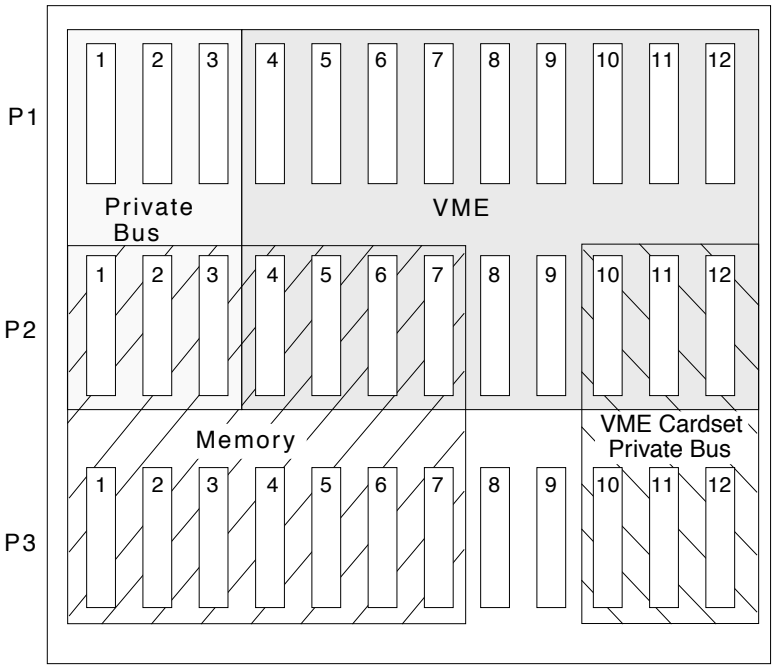
Jumper Function	Jumper Location
BG0	PX00
BG1	PX01
BG2	PX02
BG3	PX03
IACK	PX04
+5V STBY	P100

Table 10 SPARCserver 670MP Backplane Jumper Locations *(Continued)*

Jumper Function	Jumper Location
CLOCK	P10
	P11
	P12
	P13



Typical Pin/Row Layout for the Backplane Connectors



VME Bus: P1 and Row B of P2 (Slots 4-12)
Private Bus: P1 and Row B of P2 (Slots 1-3)
Memory Bus: P2 Row A and C and P3 Row B (Slots 1-7)
VME Cardset Private Bus: P2 Rows A and C and P3 Row B (Slots 10-12)

Figure 5 SPARCsystem x70 Backplane Layout (Viewed From Cabinet Rear)

Note – The preceding figure illustrates the bus type serviced by each of the 96-pin connectors in the P1, P2, and P3 areas. As shown in the detail, each 96-pin connector has an A, B, and C row. Notice the 12-slot backplane Memory Bus connects rows A and C of the P2 area and row B of the P3 connector on slots 1 through 7.

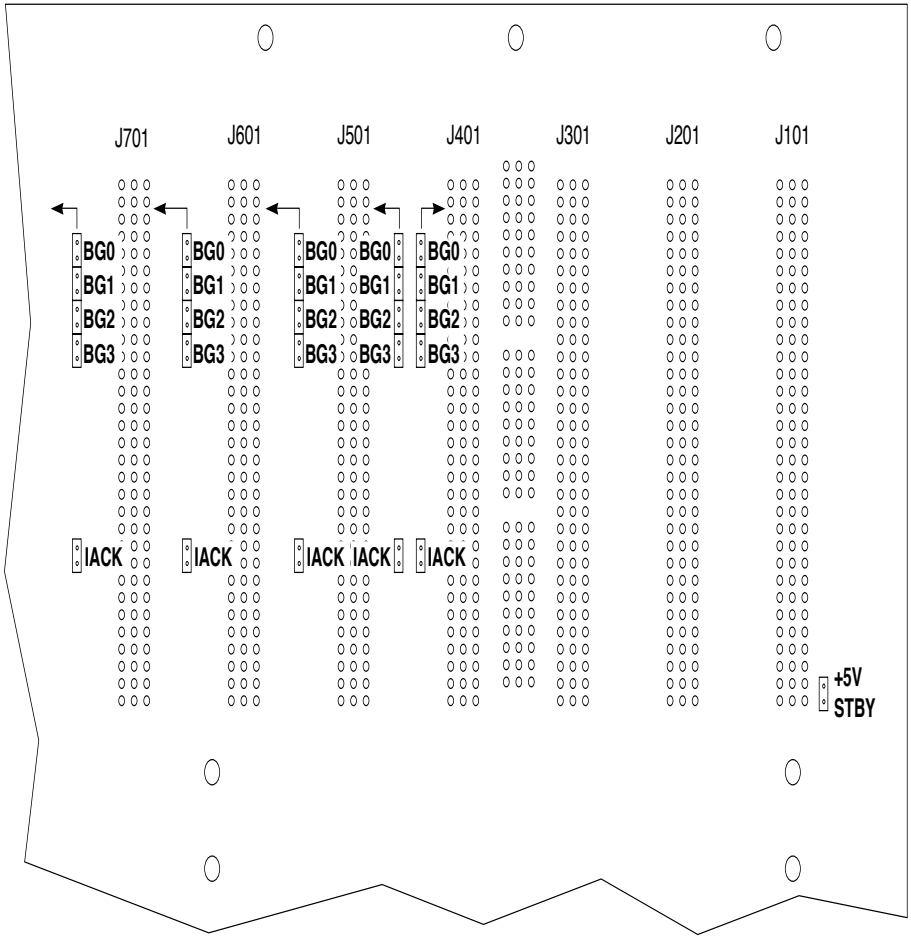


Figure 6 SPARCsystem x70 Backplane Jumper Art

SPARCserver 690MP

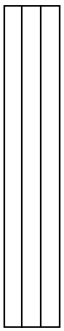
The table below describes the x90 backplane jumper functions. The second table shows the jumper locations on the backplane for each jumper function.

Table 11 SPARCserver 690MP Backplane Jumper Functions

Jumper	Slot
P4XX	4
P5XX	5
P6XX	6
P7XX	7
P8XX	8
P9XX	9
P10XX	10
P11XX	11
P12XX	12
P13XX	13
P14XX	14
P15XX	15
P16XX	16

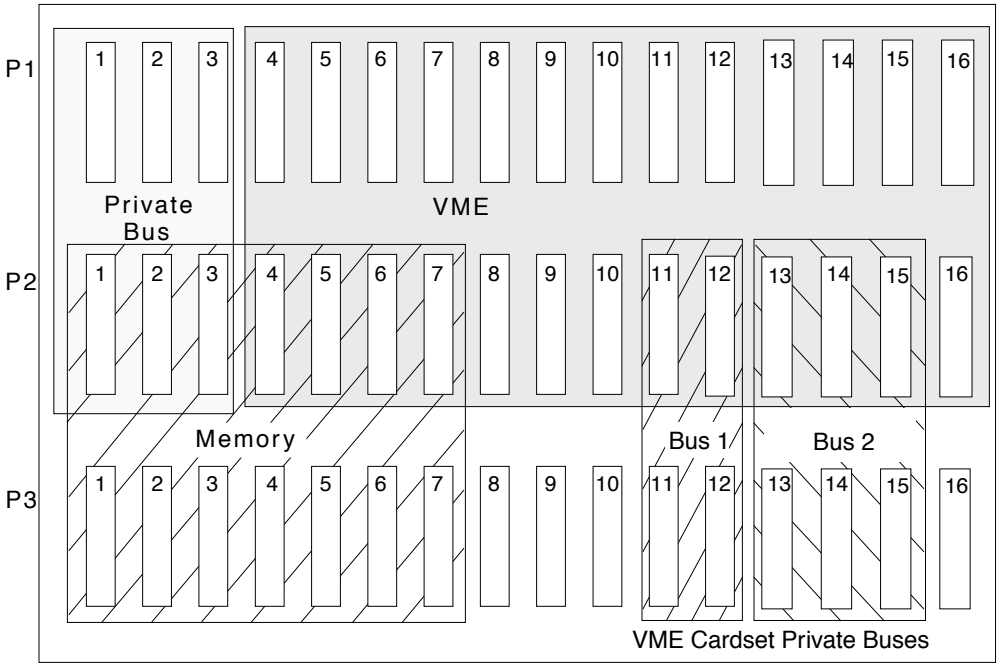
Table 12 SPARCserver 690MP Backplane Jumper Locations

Jumper Function	Jumper Location
BG0	PX00
BG1	PX01
BG2	PX02
BG3	PX03
IACK	PX04
+5V STBY	P100



A B C

Typical Pin/Row Layout for the Backplane Connectors



VME Bus: P1 and Row B of P2 (Slots 4-16)
Private Bus: P1 and Row B of P2 (Slots 1-3)

Memory Bus: P2 Row A and C and P3 Row B (Slots 1-7)
VME Cardset Private Bus: P2 Rows A and C and P3 Row B (Slots 11,12 and 13-15)

Figure 7 SPARCsystem x90 Backplane Layout (Viewed From Cabinet Rear)

Note – The preceding figure illustrates the bus type serviced by each of the 96-pin connectors in the P1, P2, and P3 areas. As shown in the detail, each 96-pin connector has an A, B, and C row. Notice the 16-slot backplane Memory Bus connects rows A and C of the P2 area and row B of the P3 connector on slots 1 through 7.

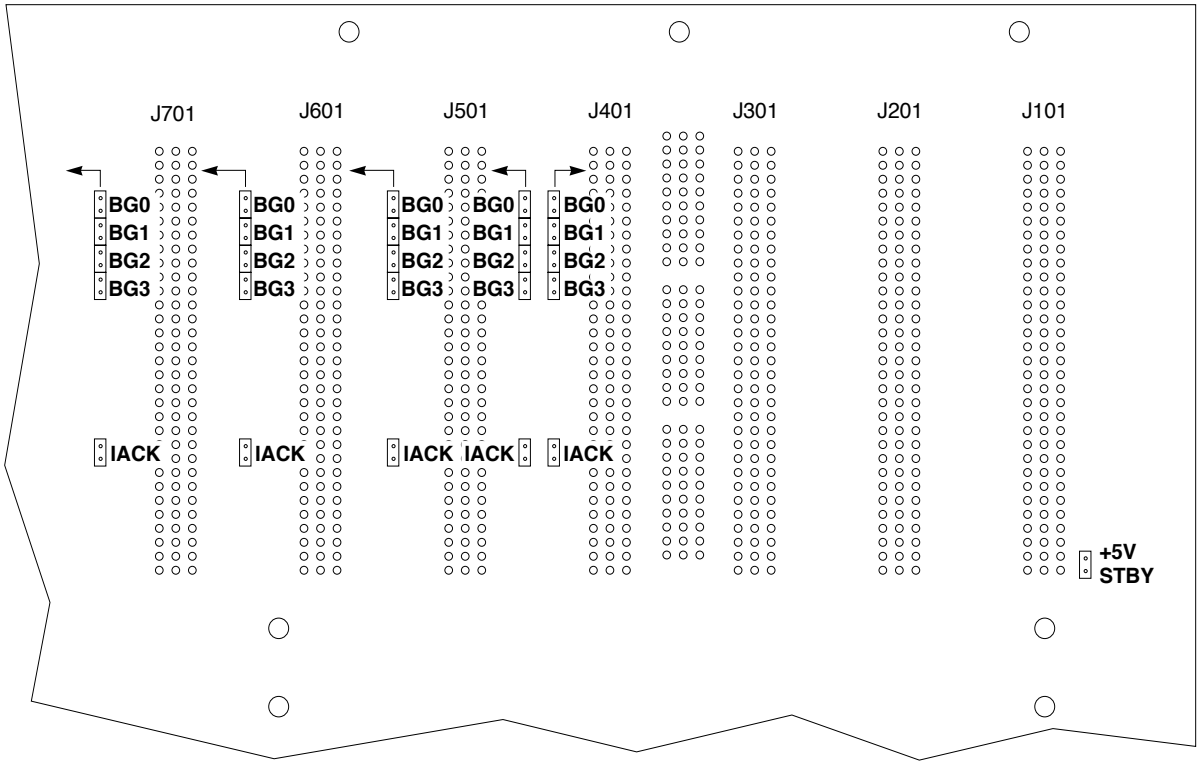


Figure 8 SPARCsystem x90 Backplane Jumper Art

690 MP ALM-2 and MCP Product Notes

A total of eight VME based communication boards (MCP and ALM-2) may reside in the system at one time. Any combination of up to four MCP and eight ALM-2 options may be configured for this system.

690MP Interrupt Vector Conflicts

The Asynchronous Line Multiplexer-2 (ALM-2) shares VME interrupt vector assignments and address space with the Multiprotocol Communication Processor (MCP). Because of these possible conflicts, and a possible physical space restriction in the Data Center Cabinet, the following must be applied when installing an ALM-2 into a cardcage that also contains MCPs.

Table 13 ALM-2 and MCP Vector Interrupt Assignments

Installed Board	Device Address (Hex)	VME Vector Interrupt Assignment
1st Board (ALM-2 or MCP)	0x01000000	8b
2nd Board (ALM-2 or MCP)	0x01010000	8a
3rd Board (ALM-2 or MCP)	0x01020000	89
4th Board (ALM-2 or MCP)	0x01030000	88
5th Board (ALM-2)	0x02000000	a0
6th Board (ALM-2)	0x02010000	a1
7th Board (ALM-2)	0x02020000	a2
8th Board (ALM-2)	0x02030000	a3

As you can see from the table, the vector interrupt assignments of the ALM-2 and the MCP are the same. This makes the following instructions necessary.

690MP Board Device Sequence

When installing the ALM-2 or MCP, the boards ***must*** be installed in proper address order. There are four VME board address positions available that can accommodate either the ALM-2 or MCP board (devices 0-3). Therefore, one address position can only accommodate one board type, and any MCP or ALM-2 must be installed in the proper board device sequence.

Table 14 Board Device Sequence

1st board (MCP or ALM-2)	Device 0
2nd board (MCP or ALM-2)	Device 1
3rd board (MCP or ALM-2)	Device 2
4th board (MCP or ALM-2)	Device 3
5th board (ALM-2)	Device 4

Table 14 Board Device Sequence *(Continued)*

1st board (MCP or ALM-2)	Device 0
6th board (ALM-2)	Device 5
7th board (ALM-2)	Device 6
8th board (ALM-2)	Device 7

Note – Refer to the specific ALM-2 or MCP Configuration Procedure for information on board device addressing.

For example, if you had two MCP boards already installed (1st and 2nd MCP boards) and you then wanted to install two ALM-2 boards, you would need to configure and install the two ALM-2 boards as the 3rd and 4th ALM-2 boards respectively.

690MP VME Address Conflicts

The ALM-2 and MCP must not be installed using identical VME addresses (board device numbers).

The ALM-2 board number (VME Address) is hardware selected on the board. If necessary, refer to the *ALM-2 Installation and Configuration Manual*, part number 813-1029, for information on setting/verifying the ALM-2 board address (board address selection is identical for the MCP).

The ALM-2 and MCP boards occupy the identical VME address space as well as interrupt vectors, and both are known to the CPU as **mcpx** (where **x** is a number **0** through **7**). So, for example, if two MCP boards are already present in the cardcage and you wish to add an ALM-2, the ALM-2 would be designated as **mcp2** in the VME addressing (with the two MCP boards being designated **mcp0** and **mcp1** respectively).

Address and Interrupt Vector Assignments

The allocation strategy for interrupt vectors and 32 bit VME addresses is summarized in the following tables.

Table 15 VMEbus Control Space

PA(35:00)	VMEbus Control Space
0xFD000000X	VMEbus Interrupt Vector Register
0xFD0000010 - 0xFDEFFFFFF	Reserved
0xFDF000000 - 0xFDF007FFF	IOC Tags
0xFDF008000 - 0xFDF00FFFF	IOC Data Diagnostic Access
0xFDF010000	VMEbus Interface Control Register
0xFDF010004	VMEbus Interface Asynchronous Error Address Register
0xFDF010008	VMEbus Interface Asynchronous Error Status Register
0xFDF01000C - 0xFDF01FFFF	Reserved
0xFDF020000 - 0xFDF027FFF	Cache Flush
0xFDF028000 - 0xFDFFFFFFF	Reserved

VMEbus Address Allocations

The following VME address space maps include many devices that are not supported on the Sun 600MP product family. These older devices are included here for historical purposes. Non-supported device addresses should be treated as reserved space.

Table 16 VME 32-Bit Address Space Layout

Start Address	Size	Description
0x00000000	8MB	DVMA
0x00800000	8MB	Reserved
0x01000000	112MB	<=2MB Sun devices
0x08000000	128MB	Sun graphic devices
0x10000000	80MB	<=2MB OEM devices
0x15000000	176MB	>2MB OEM devices
0x20000000	1536MB	>2MB Sun devices
0x80000000	1920MB	Reserved
0xF8000000	48MB	Sun-4/110 Sun devices
0xFB000000	64MB	Sun-4/110 OEM devices
0xFF000000	16320KB	Reserved for 24 bit address space
0xFFFF0000	64KB	Reserved for 16 bit address space

Table 17 VME 24-Bit Address Space Layout

Start Address	Size	Description
0x000000	1MB	DVMA
0x100000	1MB	Reserved
0x200000	2MB	<=256KB Sun devices
0x400000	4MB	>256KB Sun devices
0x800000	4MB	>256KB OEM devices
0xC00000	2MB	<=256KB OEM devices
0xE00000	1MB	Multibus to VMEbus memory space
0xF00000	960KB	Reserved
0xFF0000	64KB	Reserved for 16 bit address space

Table 18 VME 16-Bit Address Space Layout

Start Address	Size	Description
0x0000	32KB	OEM devices (allocate from low)
0x8000	32KB	Sun devices (allocate from high)

Table 19 VME Vector Layout

0x40-0x5F	Sun Disk Controllers
0x60-0x6F	Sun Tape Controllers
0x70-0x7F	Sun Network Interfaces
0x80-0x87	Sun Parallel Interfaces
0x88-0xA3	Sun Serial Interfaces
0xA4-0xA7	Sun IPC
0xA8-0xAB	Sun Frame Buffers
0xAC-0xAF	Sun Graphics Processors
0xB0-0xC7	Sun Misc.
0xC8-0xFF	OEM devices

Table 20 VME 32 -Bit Address Space Allocation

Address	Size	Data Size Slave/Master	Device Name	Device Description
0x01000000	64KB	32	mcp0	Sun Serial Line Controller (ALM-2)
0x01010000	64KB	32	mcp1	Sun Serial Line Controller (ALM-2)
0x01020000	64KB	32	mcp2	Sun Serial Line Controller (ALM-2)
0x01030000	64KB	32	mcp3	Sun Serial Line Controller (ALM-2)
0x01080000	1KB	32/32	ipi0	Sun IPI-2 Disk Controller (Panther)
0x01080400	1KB	32/32	ipi1	Sun IPI-2 Disk Controller (Panther)
0x01080800	1KB	32/32	ipi2	Sun IPI-2 Disk Controller (Panther)
0x01080c00	1KB	32/32	ipi3	Sun IPI-2 Disk Controller (Panther)
0x01081000	1KB	32/32	ipi4	Sun IPI-2 Disk Controller (Panther)
0x01600000	2MB	32	fddi0	Sun FDDI Controller
0x01800000	2MB	32	fddi1	Sun FDDI Controller
0x02000000	64KB	32	mcp4	Sun Serial Line Controller (ALM-2)
0x02010000	64KB	32	mcp5	Sun Serial Line Controller (ALM-2)
0x02020000	64KB	32	mcp6	Sun Serial Line Controller (ALM-2)
0x02030000	64KB	32	mcp7	Sun Serial Line Controller (ALM-2)
0xff800000	64K	32	pr0	Prestoserve (VME)
		32	ne0	NC-400 NFS Accelerator
		32	ne1	NC-400 NFS Accelerator
		32	ne2	NC-400 NFS Accelerator
		32	ne3	NC-400 NFS Accelerator
		32	ne4	NC-400 NFS Accelerator

Table 21 VME Interrupt Vector Allocation

Vector	Device
0x40-0x43	sc0-3, si0-3, se0-3
0x44-0x47	xdc0-3
0x48-0x4B	xyc0-3
0x4C-0x50	ipi0-3
0x51-0x5F	Reserved for Disk Controllers
0x60-0x63	tm0-3
0x64-0x67	xtc0-3
0x68-0x6F	Reserved for Tape Controllers
0x70-0x73	ec0-3
0x74-0x75	ie0-1
0x76-0x77	ie2-3
0x78-0x7B	fddi0-3
0x7C	ie4
0x7D-0x7F	Reserved for Network Interfaces
0x80-0x83	vpc0-3
0x84-0x87	vp0-3
0x88-0x8B	mti0-3, mcp3-0
0x8C-0x8F	dcp0-3
0x90-0x9F	zs0, zs1
0xA0-0xA3	Reserved for Serial Interfaces, mcp4-7
0xA4-0xA7	pc0-3
0xA8	cgtwo0
0xA9	tvone
0xAA-0xAD	vx0-3
0xAF	Reserved for Frame Buffers
0xB0-0xB3	vtx0-3

Table 21 VME Interrupt Vector Allocation *(Continued)*

Vector	Device
0xB4-0xB7	Reserved for Channel Attach
0xB8-0xB9	tbm1-0
0xBA-0xBB	hss0-1
0xBC-0xC7	Reserved for Sun
0xC8-0xFF	Reserved for OEM

Slot 1 Functions

When the 600MP slot 1 pin is IN, the 600MP enables/disables the following functions:

- the arbitration function
- the system reset
- the free-running serial clock
- the bus grant 0, 1, 2, on the VMEbus
- the beginning of the IACK* daisy chain

Note – If the Sun 600MP system board is not installed in a Sun backplane, tie IACK* to IACKIN* on slot 1 of the backplane.

Diagnostic Loopback

When the Loopback Enable Bit in the Control Register is asserted, it is possible for the processor to issue a cycle as a VMEbus Master which the VME Slave port recognizes and loops back through the I/O Cache, if appropriate, and does a DVMA access to Main Memory.

The Sun 600MP hardware provides the capability to loop back in order to self-test the area. It is extensively used by self-test and system diagnostics. Diagnostic loopback is *not* enabled during normal operations. The VME hardware is automatically exercised at boot time, and not available for other use. The address range is unique.

Note – There is no loopback jumper on the Sun 600MP. The loopback function is controlled by a write to the VMEbus Interface Control register.

Open Boot PROM dev_info

The first line in the following description,`/iommu@f,e0000000,` indicates that there is an IOMMU on the system and the control registers for it starts at PA[35:0] 0xf,e0000000.

The second line, `/iommu@f,e0000000/vme@f,df010000`, indicates that there is a VME bus that goes through the IOMMU on the system and the control registers start at PA[35:0] = 0xf,df010000.

The rest of the output (lines 3, 4, 5, 6 and 7) indicate that there are two IPI3 controllers on the system. If no IPI is present, these lines do not appear.

```
/iommu@f,e0000000

/iommu@f,e0000000/vme@f,df010000

/iommu@f,e0000000/vme@f,df010000/SUNW,pn@4d,1080000

/iommu@f,e0000000/vme@f,df010000/SUNW,pn@4d,1080000/ipi3sc@1,0

/iommu@f,e0000000/vme@f,df010000/SUNW,pn@4d,1080000/ipi3sc@1,0/id

/iommu@f,e0000000/vme@f,df010000/SUNW,pn@4d,1080000/ipi3sc@2,0

/iommu@f,e0000000/vme@f,df010000/SUNW,pn@4d,1080000/ipi3sc@2,0/id
```

Related Device Tree Nodes

Each device tree node has a ‘name’ property that helps to identify the nodes. The physical address of the registers for a node are given in a property called ‘reg’.

The property ‘address’, if present, is the virtual address(es) for the register(s) specified by the ‘reg’ property.

The property ‘page-size’ at the ‘iommu’ node indicates that IOMMU maps 0x1000 bytes or 4K at a time.

The ‘cache-coherence?’ property of the ‘iommu’ node indicates that this IOMMU is cache coherent with the rest of the system.

The ‘ranges’ properly of the ‘vme’ nodes gives the address translation from VMEBus to this machines physical address space.

For example, the following line from the ‘ranges’ property describes the translation for VMEA32D32.

```
0000004d 00000000         0000000d 00000000      ff000000
Address in                Physical address       Max. Size
VMED32A32 Supervisor      on this machine
```

The ‘intr’ property describes the interrupt level(s) and vectors used by a device such as SUNW,pn the Sun IPI3 channel adapter.

```
name = iommu
address   = ffee1000
reg   = 0000000f e0000000   00000300
page-size   = 00001000
cache-coherence?


name = vme
address   = ffedf000 ffede000
reg   =
0000000f df010000   0000000c
0000000f d0000000   0000000e
ranges   =
00000009 00000000   0000000a 00000000   ff000000
0000000d 00000000   0000000c 00000000   ff000000
00000029 00000000   0000000a ffff0000   00010000
0000002d 00000000   0000000c ffff0000   00010000
00000039 00000000   0000000a ff000000   00ff0000
0000003d 00000000   0000000c ff000000   00ff0000
00000049 00000000   0000000b 00000000   ff000000
0000004d 00000000   0000000d 00000000   ff000000
00000069 00000000   0000000b ffff0000   00010000
0000006d 00000000   0000000d ffff0000   00010000
00000079 00000000   0000000b ff000000   00ff0000
0000007d 00000000   0000000d ff000000   00ff0000
iocache?
```

```
name = SUNW,pn
reg   =
0000004d 01080000   00000400
0000004d 01080400   00000400
0000004d 01080800   00000400
0000004d 01080c00   00000400
0000004d 01081000   00000400
device_type = ipi3
intr   =
00000043 0000004c
00000043 0000004d
00000043 0000004e
00000043 0000004f
00000043 00000050


name = ipi3sc
reg   = 00000001 00000000   00000001
device_type = ipi3-slave


name = id
device_type = block


name = ipi3sc
reg   = 00000002 00000000   00000001
device_type = ipi3-slave


name = id
device_type = block
```

Sun-4m Device Driver Developer Notes

Note – The Sun-4m device driver information in this section is specific to SunOS 4.x. Solaris 2.x implements architecture specific driver information as described in *Solaris 2.0 – Writing Device Drivers*.

In general, well-behaved SBus and VME bus device drivers should migrate to a Sun-4m kernel with no or few modifications. “Well-behaved” is minimally defined as:

- The driver does not peek into a DVMA buffer without properly mapping it into the kernel’s address space.
- Only the standard mb_XXX() bus setup and tear-down routines are used to effect mappings between kernel and device address spaces.
- The driver does not modify kernel page table entries or mappings, nor does it call the hat_XXX() or segkmem_XXX() routines.

The primary difference between the 600MP and earlier Sun-4 systems (from the device driver implementor’s perspective) is the introduction of an I/O MMU (IOMMU). Details on managing IOMMU mappings are provided in the next section. The 600MP has an I/O cache (IOC) which is essentially the same cache present on the SPARCserver 490. There are also write buffers between the MBus and the SBus/VME bus interfaces, described in the next section.

I/O cache dependencies should be conditionally compiled using the `IOC` constant:

```
#ifdef IOC
#endif IOC
```

Code included to support the `Sun-4m` architecture, which may include IOC or IOMMU setup, should be surrounded with `Sun-4m` conditional compilation directives:

```
#ifdef sun4m
...600MP specific code
#ifdef sun4m && IOC
...600MP I/O cache code
#endif sun4m && IOC
#endif sun4m
```

In general, device drivers that use the default SBus and VME I/O mappings should remain portable. Drivers that create mappings larger than 1 Mbyte will be accelerated by the larger SBus and VME I/O maps available on the Sun-4m platform, but using the larger VME map may make the driver less portable. Mappings, I/O space and IOC interactions are discussed in the following section.

Although the interrupt steering and trap handling is invisible to most device drivers, care should be taken not to call routines that change the CPU priority level as a side effect. A device that interrupts at SBus priority X and was mapped to IRL Y on a Sun-4 or Sun-4c machine will now have its interrupt mapped to IRL Z on the Sun-4m architecture. Z may be greater than Y, since both VME and SBus interrupt levels are "merged" in the Sun-4m SPARC IRL mappings.

Calling a kernel service routine that changes the CPU priority to a level below the current one may result in corrupted data structures or a variety of data fault panics. The following guidelines should be observed:

- Networking and `mbuf` utilities may be called while handling a device interrupt at SBus or VME level 4 and below.
- STREAMS utilities may be called while handling a device interrupt at SBus or VME level 5 and below.
- Networking and `mbuf` utilities should not be called while the CPU priority is above IRL 7 (`splimp`).
- STREAMS utilities should not be called with the CPU priority above IRL 10 (`splstr`).

More detail may be found in the *Writing Device Drivers Addendum*.

The Sun-4m `user` and `proc` structures are larger than the equivalent Sun-4 or Sun-4c structures. New fields have been added to handle multiple processors and the SRMMU. Drivers that reference `user` or `proc` structures must be recompiled on a Sun-4m machine because offsets into these structures are machine architecture-specific.

IOMMU and I/O Cache (Software View)

In previous Sun-4 and Sun-4c machines running SunOS 4.x, the DVMA space was carved out of the kernel's address space, and DVMA operations shared the single MMU with the CPU. The Sun-4m IOMMU separates the I/O address space from the kernel's memory mappings. Memory, processors and the SBus interface reside on the MBus. Each non-memory module on the MBus requires an MMU to translate virtual to physical addresses: the CPU uses the SRMMU, and the SBus uses the IOMMU. VME bus access is handling through the SBus/VME bus interface, so the same SBus-MBus interconnection rules apply.

Separating the DVMA and CPU address mappings makes DVMA addresses valid only from the point of view of a device on the SBus or VME bus. The CPU (and therefore the kernel) cannot interpret DVMA addresses that are mapped through the IOMMU; the CPU only goes through the SRMMU to access memory on the MBus. The converse is also true: valid kernel memory mappings are not necessarily also present in the IOMMU, so kernel addresses are (in general) invalid for DVMA.

Some of the default I/O mappings are double-mapped using both the SRMMU and IOMMU, allowing devices and the kernel to access the same underlying physical memory. This section discusses the Sun-4m DVMA memory maps, kernel facilities for their management, and the Sun-4m I/O cache.

IOMMU Mappings

It helps to think of the IOMMU as a gateway between devices (SBus or VME) and the MBus. All DVMA masters share the single IOMMU, which supports DVMA spaces of 16Mbytes - 2 Gbytes. The default DVMA space on the 600MP series is 16 Mbytes, divided into 4 DVMA maps:

`sbusmap`	Default for SBus devices, about 1 MB
`bigsbusmap`	8 MB map for SBus devices
`vme24map`	Default map for VME devices, about 1 MB
`vme32map`	32-bit VME map, about 6 MB

32-bit VME devices may be accessed through the `vme24map`; the map name is more descriptive of the addressing range of the map rather than the devices on it. Only 32-bit devices may be mapped in the `vme32map`. Most devices will use the default maps, which retain their names from other platforms. Using the default maps ensures compatibility across Sun-4, Sun-4c and Sun-4m platforms: SBus and VME devices will continue to use `dvmamap` and `mb_hd.mh_map` (respectively) as arguments to `mb_XXX()` routines.

Use of the other two maps may improve performance on the 600MP series at the slight expense of reduced portability. The `vme32map` allows large VME bus transfers to be performed more efficiently, with fewer map allocations and

releases. Similarly, the `bigsbusmap` may be used by SBus devices with large device address spaces. The larger DVMA space allows the device to be mapped in its entirety instead of forcing the driver to manage segments of the mapping.

The `vme24map` is visible from the kernel: it is the only map that can be accessed through the SRMMU. The SBus maps and `vme32map` are only accessible through the IOMMU. The figure on the next page shows the complete IOMMU memory map for the 600MP series. Note that the VME A24 device space starts at offset `0xff800000`, which is not the same offset used by other Sun-4 platforms.

The IOBP areas are double-mapped (by default) into CPU and device address spaces, since the IOBP buffers are frequently used by both driver code and the devices themselves. Regions of the `vme24map` are double-mapped into CPU address space as well when the DVMA mapping is created via the `mb_XXX`() routines.

Table 22 IOMMU Virtual Address Map

ff000000	bigsbusmap start	8 MB
ff7fffff	bigsbusmap end	
ff800000	IOBP map for VME	2 pages
ff802000	vme24map start	1 MB
ff900000	vme24map end	
	vme32map start	6 MB
ffeffffff	vme32map end	
fff00000	IOBP map for SBus	2 pages
fff02000	sbusmap	1 MB
ffffffff	end of sbusmap	

Managing IOMMU Translations

IOMMU mappings are not synchronized with the SRMMU. Therefore, it is imperative that drivers to not interleave CPU and device accesses to memory without first calling one of the mb_XXX() setup routines. IOMMU translations may be created once at device initialization, or managed dynamically using `mbsetup`() and `mbrelease`().

The IOBP maps occupy the first two pages of the default SBus and VME bus DVMA maps, leaving slightly less than 1 MB for driver-created mappings. IOPB areas are double mapped into the IOMMU and kernel's address space, since they are frequently used for command blocks and must be accessed by both drivers and devices.

Three new flags have been added to `sundev/mbvar.h` to indicate that a non-default DVMA map should be used, or that the mapping should be made in both the IOMMU and the SRMMU. The following table summarizes the map names, flags to pass to the mb_XXX() routines, and the addresses used by the kernel and device.

Table 23 Map Names, mb Flags, Kernel and Device Addresses

Mapping	**Map Name**	**Flags**	**Map Used**
SBus IOBP	iopbmap		iopbmap
VME IOBP	iopbmap		iopbmap
VME A24, < 1M	md_mh.mh_map		vme24map
VME A32, < 1M	md_mh.mh_map		vme24map
VME A32, < 1M	md_mh.mh_map	MDR_DVMA_PEEK	vme24map
VME A32, > 1M	md_mh.mh_map	MDR_VME32	vme32map
SBus, < 1M	dvmamap		sbusmap
SBus, > 1M	dvmamap	MDR_BIGSBUS	bigsbusmap

Some additional notes on IOMMU mappings:

- The IOBP address passed to a VME device should have the offset of the DVMA area subtracted from it, but SBus IOBPs do not require this translation:

```
vme_dev_address = iopbmem - DVMA;
sbus_dev_address = iopbmem;
```

- DVMA space for VME A24 devices as well as VME A32 devices that use the `MDR_DVMA_PEEK` flag will be double-mapped. No mapping created in the vme32map can be seen by the kernel: these addresses are only for use by the VME devices.
- Device drivers can access the DVMA area using `DVMA`[] if the `MDR_DVMA_PEEK` flag is used. If a drive allocates a mapping with both the `MDR_DVMA_PEEK` and `MDR_VME32` flags, the `vme24map` will be used instead of the `vme32map`.
- A VME A32 device can use the vme32map and access the DVMA area after mapping it into the kernel's address space using bp_mapin() and bp_mapout(). In general, any driver that peeks into the DVMA area should be modified to use `bp_mapin`() and `bp_mapout`() instead.

  None of the SBus maps can be seen from the kernel. When the regular SBus DVMA map is used, the address passed to the SBus device should have the DVMA offset added to it; when the `bigsbusmap` is used, no DVMA offset is required:

```
dev_address = mb_mapalloc(dvmamap,MDR_BIGSBUS...);
dev_address = mb_mapalloc(dvmamap,...) +DVMA;
```

Using the larger SBus and VME DVMA maps may make drivers more efficient, since they will not be competing with other devices for the smaller 1 MB DVMA spaces. Disk transfers using the onboard SCSI host adaptor, for example, create mappings in the 1 MB `sbusmap`, so using the `bigsbusmap` for other SBus device will reduce contention for the small DVMA space. However, using the larger SBus map makes the driver nonportable.

I/O Cache

The IOC on the 600MP series is a write-back cache that is coherent with main memory on a 32-byte burst basis. Only transfers with a VME bus master and either MBus (memory) or SBus slave are cached. The IOC does not affect SBus drivers. The `mb_XXX`() routines generally manage the IOC such that it is

transparent to the driver. However, some knowledge of the underlying cache structure and management policies is useful for optimizing VME device drivers.

The IOC is organized as 1024 32-byte lines that act more like write buffers than a direct mapped cache. Each 32-byte line corresponds to a single 8k page in the DVMA space. Unlike the CPU's virtual address cache, consecutive 32-byte chunks of data do *not* fill consecutive lines in the cache. Consecutive 32-byte blocks of data re-use the same line in the IOC; consecutive 8k pages use consecutive lines. The tag and index bits simply come from different parts of the address as shown below.

:

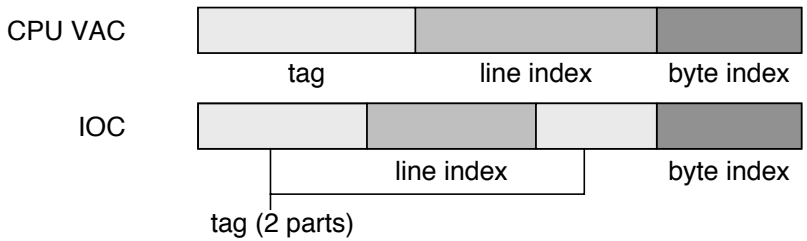


Figure 9 I/O Cache Tag and Index Bits

The IOC cache model allows an entire 8k I/O page to be cached in a single line, with repeated line fills and flushes or write backs to memory or the VME device. This is known as the "streaming I/O model."

The 600MP I/O cache is always enabled; that is, all transfers will use the cache if they can. On the 600MP system, all writes (from memory to device) are cached, since the IOC can load padding bytes to round the transfer up to a 32-byte boundary. For reads, however, the operation will only be cached if the buffer is aligned on a 32-byte boundary and the transfer size is a multiple of the line size as well. Again, the `mb_XXX`() setup routines will determine if an operation can be cached, and enable or disable the IOC appropriately for the mapping created.

Because of the asymmetry in read and write caching, it is critical that the same 8k I/O page not be used for both reading and writing without a corresponding IOMMU setup and teardown or an explicit IOC flush. Attempting to read and write from the same page may cause cache consistency problems because writes are always cached, and the cache line may include padding bytes used to align the write buffer to a 32-byte boundary.

Normally, the IOC flush is done via a call to `ioc_flush`() by the mb_XXX() teardown routines. However, a driver can explicitly call `ioc_flush`() with the IOC line number to be flushed as an argument. If a transfer spans more than one 8k page, `ioc_flush`() must be called for each page: this corresponds to flushing each cache line used. Drivers ported from other Sun machines with I/O caches should be recompiled if they call `ioc_flush`(), since this is a macro that is kernel architecture-specific.

Write buffers sit between the MBus and the SBus/VME bus interfaces. The SunOS kernel flushes these buffers before calling a device driver interrupt routine, and any attempt to read from a cached address will cause the write buffer to be flushed as well. Consider this simple sequence of events:

```
Write to device
Delay 100 usec
Write to device
```

Relative time-ordering of write operations is not preserved with the write buffers: the execution order is maintained, but the write operations may be performed back-to-back rather than with a delay between them. The actual delay may be less than 100 usec if no cache flushes occur within the delay window. If your driver relies on certain minimum inter-write time delays, you can perform a read-back from the device to force the write buffers to be flushed.

Porting Loadable Device Drivers to Sun-4m

Note – The changes pertain to both “device drivers” and “network interfaces” as referenced by the type identifiers `VDMAGIC_DRV` and `VDMAGIC_NET`

Note – The Sun-4m device driver information in this section is specific to SunOS 4.x. Solaris 2.x implements architecture specific driver information as described in *Solaris 2.0 – Writing Device Drivers*.

Because the Sun-4m/6XX has both an SBus and VME bus, some adjustments were made to the vd driver to handle the fact that both mb and devinfo style device drivers may be loaded onto the system. Existing devinfo style drivers (SBus) should compile and run without modification on a Sun-4m machine. In

most cases a recompile may not even be necessary. However, existing mb style drivers (VME) will have to be modified to be loaded properly via the vd driver on a Sun-4m. Reference <sun/vddrv.h> for specific changes.

Two new type identifiers were added so that the system can distinguish which style of driver you are loading. Devinfo style drivers still use the `VDMAGIC_DRV/VDMAGIC_NET` id's. On Sun-4m, mb style drivers will need to be modified to use the new `VDMAGIC_MBDRV/VDMAGIC_MBNET` id's to inform the system that a mb style driver is to be loaded.

Also, the format of the vdldrv/vdlnet structures are different on Sun-4m. The structure members were rearranged to allow existing devinfo style drivers to run/compile without modification. However, existing mb style drivers will have to change their vdldrv/vdlnet declaration to conform to the modified structures. Aside from the ordering of structure elements the only difference is that a "dummy" (NULL) entry must be added for the "struct dev_ops" pointer, which is used only for devinfo style drivers.

The modstat(8) utility has been modified to recognize the new driver types. It will print "Mdrv" and "Mnet" in its type field to represent `VDMAGIC_MBDRV` and `VDMAGIC_MBNET` drivers, respectively. All other modstat output will be as it has been previously.

Compatibility

Some drivers can run on 600MP without recompilation, but some drivers may require modifications. The following is a list of points to consider to determine if a existing driver is portable to 600MP:

1. User and proc structures are different on the 600MP due to the SRMMU. If your driver looks at the user or proc structures (to get pid, or uid, or proc pointers) it must be recompiled because the offsets of these fields have changed on the 600MP.

2. Does the driver use only "standard" driver supporting routines provided by the kernel, i.e. mb_mapalloc/mb_mapfree (and their mb_XXX variations) to manage DVMA mappings? (Note: segkmem_XXX, hat_XXX are not considered as standard driver supporting routines.) If your driver uses "non-standard" routines to setup DVMA mappings, while it may works on other machines but most likely it will NOT work on 600MP since mb_XXX

routines are the only routines that set up DVMA mappings on the IOMMU. These drivers MUST be converted over to use the 'standard' mb_XXX routines if they are to be portable.

3. Does the driver use DVMA[some_offset] to reference/modify data inside the DVMA data transfer buffers? If the answer is yes to this question, then:

   a. if it is a VME device, it'll run as is. However, it is advised that the driver should use bp_mapin() and bp_mapout() routines for better portability.

   b. if it is SBus device, you need to convert it to use bp_mapin()/bp_mapout(). Otherwise, the driver will pick up random values.

   This is due to the fact that DVMA addresses are now independent of the host SRMMU virtual address. For compatibility reason, the DVMA mappings in vme24map (default map for vme devices) is set up such that DVMA[] reference would still work. However, the same thing can not be done on the SBus maps, so it is no longer compatible with other Sun machines.

   c. Does your VME device run with IOC off?

   IOC only works on VME devices. IOC is turned on/off, flushed automatically by the mb_mapalloc()/mb_mapfree(). Currently, there is no easy way for a driver to disable IOC. However, the ioc_flush() routine remains the same for drivers that need to flush IOC.

Other than the points listed above, a driver should be written exactly the same way as it were written for other machines. A driver written for 600MP should work for the other machines but not necessarily vice versa since the rule for 600MP is more restrictive. The only exception to this rule is if your driver uses non-default large DVMA maps which are not supported on other machines (see *Performance Tuning* below).

4. Performance Tuning

   To optimize the performance for drivers running on 600MP, drivers can do the following:

   a. Vme32 devices: if the driver does not peek inside the DVMA data buffers without bp_mapin()/bp_mapout(), it may set the `MDR_VME32` flag to use the much larger vme32map, instead of the smaller default vme24map. If

the DVMA request size is not larger than 1M, adding this flag should not cause problems in terms of portability. It will be simply ignored by machines that do not have this map.

b. SBus devices: can set '`MDR_BIGSBUSMAP`' flag to use the big 8M map. If it uses the big map, it must NOT add the DVMA base to form the DVMA address passed to the device. Mb_mapalloc() already returns a correct 'ready to use' DVMA address. Usage of this map makes the driver NON-portable.

Note – Driver should not use vme32map or bigSBusmap as arguments to the mb_XXX routines. Instead, use flags as described.

---

5. IOMMU Bypass Mode

   If a driver uses IOMMU bypass mode, it will be responsible for its own DVMA mappings. The standard DVMA supporting routines described above will not be useful for them.

Interrupts

Hardware interrupt levels for Sun-4m are different than for Sun-4 or Sun-4c. Refer to the Sun-4m System Architecture document for details.

The Sun-4m mapping of SBus and VMEbus interrupt levels to SPARC interrupt request level (IRL) is different from previous Sun-4 and Sun-4c architectures. A device which interrupts at SBus level x which is mapped by the onboard interrupt logic to SPARC IRL y on Sun-4 or Sun-4c will now be mapped to SPARC IRL z on Sun-4m and z may be greater than y. This may introduce bugs which typically manifest themselves as corrupted data structures leading to kernel crashes.

For Sun-4m, device drivers should not make Networking or STREAMS framework function calls while operating at SPARC IRL levels higher than IRL 7 (splimp) for networking, or IRL 10 (splstr) for STREAMS. Doing so circumvents the interrupt masking being done by the networking and STREAMS subsystems themselves and risks data structure corruption.

Device drivers should do minimal processing at high interrupt levels and schedule a software interrupt for further interrupt processing, including interacting with other portions of the kernel. For Sun-4m, it is possible to call

networking and mbuf utility routines while servicing a device interrupt at VME and SBus levels 4 and below. It is possible to call STREAMS utility routines) at VME and SBus levels 5 and below.

In general, device drivers which support multiple devices which interrupt at more than one hardware interrupt level must take precautions to service only those interrupts at the "current" interrupt priority level (via spltoipl()) in order to avoid race conditions which result in the error message "Level XXX BBB interrupt not serviced".

Write Buffers

Write buffers are used to accelerate writes and reduce bus occupancy for better overall system performance. Write buffers exist both for programmed I/O and DVMA activity. Use of the mb_xxx() routines guarantees correct operation. All write buffers in the Sun-4m architecture follow these rules:

1. Once a write buffer has accepted a write, it must either guarantee that the write can occur without error, or the write buffer is responsible for reporting those errors.

2. Write buffers are read-stall; that is, after a write buffer has accepted a write, any subsequent access to that device must wait for the write operation to complete (order is maintained). Although write buffers are not visible to device drivers their effect may not be. While order is maintained, the relative timing of writes to the device may be significantly different from the issuing (CPU) timing.

SBus Slot Configuration Register

There is an SBus slot configuration register for each SBus slot in the system. Each SBus slot configuration register provides information about the slave device in that slot (slave support for 64-, 32-, 16-, and 8-byte bursts), and is also used for IOMMU bypass management for that slot. The boot code is expected to configure the slot based upon FCodes associated with the SBus device. Refer to the *Open Boot PROM 2.0 Toolkit Reference Guide*, P/N 800-6076-xx. Failure of the device firmware FCode to support this property may result in less than optimal slave access performance for the device as only 4-byte word sized slave transfers will be used.

Open Boot PROM

SPARCsystem 600MP is released with version 2 of the open boot PROM firmware. Refer to the *Open Boot PROM 2.0 Toolkit Reference Guide*, P/N 800-6076-xx for details.

Questions and Answers

Question:

What VMEbus restrictions are there on using two 600MP system boards in the same backplane?

Answer:

If you have two 600MP system boards in the same backplane, talking over the VMEbus, and they both respond to the same address, you will have a problem. For example, system board A wants to talk to address 0 on system board B, and system board B wants to talk to address 0 on system board A. It is very important that loopback is disabled so that they don't recognize their own address. VME loopback is disabled through the VMEbus Interface Control Register.

Question:

Is there anything special in the Sun VME interrupt area?

Answer:

No, you should jumper disable as usual. All interrupts are 2-cycle synchronized before encoding, so there is no chance of spurious interrupts. Also, the 600MP systems are VME rev C.1 instead of B, so the IACK qualification actually makes the IACK cycle more bomb-proof.

Question:

If another device is VMEbus master and the Sun processor tries to become master of the VMEbus. How long will the Sun processor wait for access to the bus before timing out?"

Answer:

The Sun system board will wait forever. The VME master time-out is based upon the time that the VME master P1_AS* being asserted; the clock doesn't start until the board gets a grant. Since there is no guarantee that the 600MP system board is the arbiter, there is no guarantee that you'll get access to the bus in a deterministic time (this is if the 600MP system board is not in slot 1, arbiter position).Once the 600MP system board is granted the bus, the timeout is > 200uSec.

600MP Series Documentation List

Table 24 Supporting Hardware and Software Documentation *(1 of 3)*

Manual Title	Part Number
System and Expansion Enclosure Installation	
SPARCsystem 630MP Installation Manual	800-5937-XX
SPARCsystem 670MP Installation Manual	800-5904-XX
SPARCserver 690MP Installation Manual	800-5935-XX
SCSI Expansion Pedestal Installation Manual	800-5905-XX
Expansion Cabinet Installation Manual	800-5936-XX
Service Manuals	
600MP System Board and Expansion Memory Installation and Service Manual	800-5318-XX
5-Slot Office Pedestal Service Manual	800-3272-XX
12-Slot Office Pedestal Service Manual	800-3255-XX
56″ Data Center Cabinet Service Manual	800-3259-XX
56″ Expansion Cabinet Service Manual	800-6371-XX
Site Preparation	
Sun Microsystems Site Preparation Guide	800-5215-XX
Site Preparation Sticker/Board Set	800-6220-XX

Table 24 Supporting Hardware and Software Documentation *(2 of 3)*

Manual Title	Part Number
Site Preparation Option Specification	800-3270-XX
SPARC Module Installation	
SM100 SPARC Module Installation Guide	800-6739-XX
VME Board Installation	
ISP80 Disk Controller Board Installation Manual	800-1050-XX
ALM-2 Installation Manual	813-1029-XX
Sun-4m VME Prestoserve Installation Manual	800-6608-XX
Sun VME Network Coprocessor Installation Manual	800-6881-XX
FDDI/DX Installation Manual	813-1053-XX
SBus Card Manuals	
SBE/S SBus Card User's Installation Guide	800-6475-XX
SBus Card Installation Guide for Deskside and Data Center Cabinet Systems	800-6366-XX
Prestoserve SBus User's Guide	800-6396-XX
Unbundled Diagnostics Manuals	
ScanTool User's Guide	800-5865-XX
Open Boot PROM 2.0 Toolkit User's Guide	800-5674-XX
Open Boot PROM 2.0 Toolkit Reference Guide	800-6076-XX
DiagPROM User's Guide	800-5862-XX
Using the Exec	800-6283-XX
Unbundled Diagnostics Manuals	
Basic System Diagnostics	800-6284-XX
Network Diagnostics	800-6285-XX
Graphics Diagnostics	800-6286-XX
Peripherals Diagnostics	800-6287-XX
SunDiagnostics Executive 2.0 Quick Reference Guide	800-6288-XX
SunDiagnostics Executive 2.0 Advanced User's Guide	800-6289-XX

Table 24 Supporting Hardware and Software Documentation *(3 of 3)*

Manual Title	Part Number
SunDiag 2.2 User's Guide	800-6020-XX
ScanTool Advanced User's Guide	800-5866-XX
Regulatory Manuals	
Pedestal Regulatory Compliance Manual	813-1105-XX
Data Center Regulatory Compliance Manual	813-2100-XX
Option Installation Notes	
SunCD Installation Note	800-5948-XX
150MB Tape Installation Note	800-5946-XX
1.3GB SCSI Disk Installation Note	800-6227-XX
2.3GB 8mm Tape Installation Note	800-5944-XX
Front Load Tape Installation Note	800-5945-XX
SIMM Option X168A Installation Note	800-6594-XX
600MP Boot PROM Replacement Part Installation Note	800-6614-XX
600MP NVRAM Replacement Part Installation Note	800-6620-XX
600MP Expansion Memory Board Installation Note	800-6621-XX
Installing the Desktop Storage Pack to the SPARCserver 600 Series	800-6944-XX
Installing the Desktop Storage Module to the SPARCserver 600 Series	800-6945-XX
Upgrade Procedures	
630MP Upgrade Installation Instructions	800-6075-XX
670MP and 690MP Upgrade Installation Instructions	800-6074-XX
X/260 or X/280 to 670MP or 690MP Upgrade Installation Instructions	800-6611-XX

Revision History

Revision	Dash	Date	Comments
800-6738-10	-10	December 1991	First Customer Ship (FCS)
800-6738-11	-11	August 1992	2nd Release